Vocal Source Separation Using Local Attention and Temporal Convolutional Networks in Both Waveform and Spectrogram

Jiabin Wang
Department of Computer Science
Stanford University
dovejb@stanford.edu

Abstract

Vocal Source Separation is a subtask of Music Source Separation or Speech Separation. As sequence data, many approaches using RNN based networks to retrieve audio features like Hybrid approach(1) and SWave(2). Like RNNs, transformers and attention mechanisms are proved to work well on sequence data, but they can process in parallel thus faster. Temporal convolutional network (TCN) is a framework which employs casual convolutions and dilations so that it is adaptive for sequential data, and because it's a CNN based network, it can process in parallel as well. Conv-Tasnet(3) uses TCN for mask estimation for Source Separation in time domain. There are more approaches only learn from time-domain data, and some approaches use a hybrid method(1). We propose a new architecture including Local Self Attention, TCN and the the hybrid of waveform and spectrogram for this separation task.

1 Introduction

Music Source Separation or Speech Separation is to separation multiple audio sources from the mixture signal. By vocal source separation, we can get the vocal source for main melody analysis, and the instrumental part can work for producing cover versions of any song which the official accompanies is not provided. The speech separation is a basic task with a wide range of applications, including mobile communication, speaker recognition.
What interests me the most is art generation, especially music and novel. There are many toy solutions, generating non-sense materials, because there are not so much data to learn from. But for music generation, I think vocal source separation is an important basic task, because we can extract main melodies from any existing and future songs with it, and we can either generate directly from them, or apply a music transcript before generating. Sufficient data is essential for good generation.
The input to our algorithm is an audio segment. We then use a neural network to output the vocal source and accompaniments of it with a same length.

2 Related work

We can group the previous works into these categories. (1) Data formats, most of works directly process audio data in time-domain representation, such as Att-Tasnet (4), Conv-Tasnet(3), Swave(2). Some of them use a hybrid of time-domain and frequency-domain representation, such as Hybrid Demucs(1) and Hybrid Transformers(5). There are also approaches using STDCT(6) and other

formats of audio data. (2) Encoder algorithms, there are CNNs[Conv-Tasnet(3), Swave(2)], and Attention based encoders[Att-Tasnet (4), (5)], and the mixture of CNN/RNN/Attention[Hybrid Demucs(1)]. (3) Separation Network, there are UNet[Hyprid Demucs(1)], TCN[Conv-Tasnet(3), Att-Tasnet (4)] and LSTM[Swave(2)].
Some of the related work are for speech separation, but vocal source separation from music is different because music has pitch, and the pitch range and timbre of humans are obviously different from instruments. Spectrogram is the most intuitive representation of pitch, so we should take this into consideration to gain the potential to reach the theoretical limit, so the previous works which didn't use spectrogram are not perfect for music source separation. Some of the work use RNNs based architectures, RNNs are good for sequence data, but they are slow and require large amount of memory, so they cost more hardware to extend, more time-consuming to train, and this also happens at runtime.
But some of these work also solve these pitfalls partially, Conv-Tasnet(3) and Att-Tasnet(4) uses TCN for causal sequence data estimation, Hybrid Demucs(1) involves both time-domain and frequency-domain. We make use of these advantages to build a new architecture for vocal source separation.
Novelty: the encoder layer, separation layer, decoder layer are based on algorithm in Att-Tasnet(4). Using different audio embeddings(Waveform and Spectrogram) to compare are my own work. Also, I propose a receptive field(local size) finder method, it can decide the best local size within just 1 round of training. And a 2-step method to align the output scale to the input audio.

3 Dataset and Features

The MUSDB18 is a dataset of 150 full lengths music tracks (around 10h duration) of different genres along with their isolated drums, bass, vocals and others stems. The dataset is split into training and test sets with 100 and 50 songs, respectively. All signals are stereophonic and encoded at 44.1kHz.

- Preprocessing
  1. Select the mixture track as X, and the vocal track as Y
  2. Convert from stereo to mono
  3. Resample from 44100Hz to 16000Hz
  4. Save the X and Y to disk
- Normalization
  Wav file is normalized once it's loaded.
- Augmentation
  Apply one or more of the following augmentation methods with 50% probability.
  1. Pitch Shift (-2 to +2)
  2. Time stretch (-10% to 10%)
  3. Filter (band pass filter)
  4. Spectrogram mask (mask a rectangle range of spectrogram to zero)
- Segmentation
  A segment is 1s long(an array with size 16000). We won't segment the audio when preprocess, but load each song entirely, and random pick a start point uniformly, and with a specified length, the segment is ready. A random seed is set to make the dataset stable. This can save space and time to load data, and has an augmentation effect of time shift.
- Spectrogram
  STFT if applied to generate spectrogram, n_fft is 512, hop_size is 160, a segment [1,16000] is padded to ensure the result is [1, 257, 102, 2] (with return_complex=False, because normalization layers cannot apply on complex number). The 1st line of frequency bin and the 1st and last lines in time series is removed, the shape is now [1, 256, 100, 2], and the 2nd and 4rd dimensions are combined into one dimension, the result is [1, 512, 200].

4 Methods

1. Basic Architecture
   Figure 1 illustrates the single architecture of our algorithm(N is batch size, L is segment

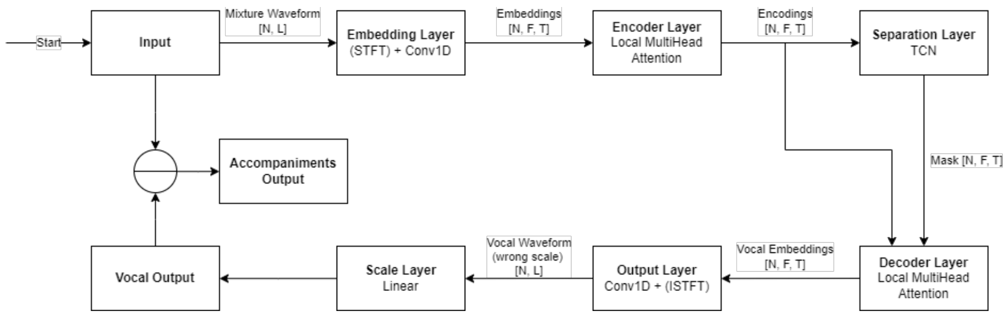


Figure 1: Single Architecture

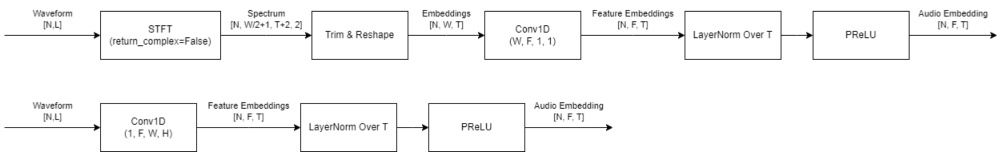


Figure 2: Spectrogram Embedding Layer and Waveform Embedding Layer

length, F is feature dimension, T is time series length). The waveform inputs firstly go through the embedding layer, and are converted to audio embeddings with shape [N, F, T], then a local multi-head attention encoder will encode the embeddings to the same shape encodings. The encodings are inputted to separation layer, the output is the mask. Then, the local multi-head attention decoder will accept both the encodings and mask, and output the predicted embeddings of vocal source. The output here may sound good, but the loudness is likely to be different to the original vocal source, so we put it into a simple scale layer. And then we get the vocal output. The input mixture subtracts the vocal output is the accompaniments output.

2. Embedding Layer
   Embedding Layer convert [N, L] waveform inputs to [N, F, T] audio embeddings. There are 2 methods: spectrogram and waveform as Figure 2 shows. The Conv1D parameters are (in_channels, out_channels, kernel_size, stride).

3. Encoder Layer
   A one layer multi-head attention encodes the embeddings with a relu activation follows. The output will perform an element-wise multiplication with the embeddings, Figure 3.

4. Separation Layer
   Then encodings are then passed to TCN Separation layer. One round of TCN have D blocks, the dilation is 1, 2, .. 2**(D-1), and these TCN blocks will repeat R times. Then the results go to Separator Conv1D after PReLU activation. The output of TCN layer is the mask of vocal source. See Figure 4

5. Decoder Layer
   Encodings and mask are the inputs of decoder layer. The mask passes through a multi-head attention block, and performs an element-wise multiplication with encodings. After that is a pointwise 1D convolution. Figure 5.

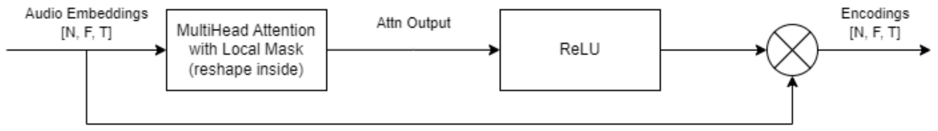


Figure 3: Encoder Layer

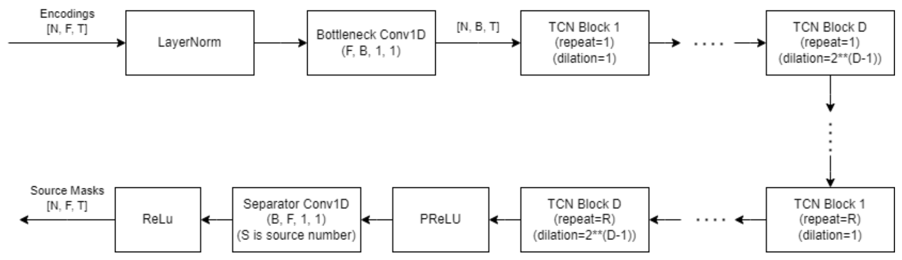

Encodings [N, F, T]
LayerNorm
Bottleneck Conv1D (F, B, 1, 1)
[N, B, T]
TCN Block 1 (repeat=1) (dilation=1)
TCN Block D (repeat=1) (dilation=2**(D-1))
TCN Block 1 (repeat=R) (dilation=1)
TCN Block D (repeat=R) (dilation=2**(D-1))
PReLU
Separator Conv1D (B, F, 1, 1) (S is source number)
ReLu
Source Masks [N, F, T]

Figure 4: TCN Separation Layer

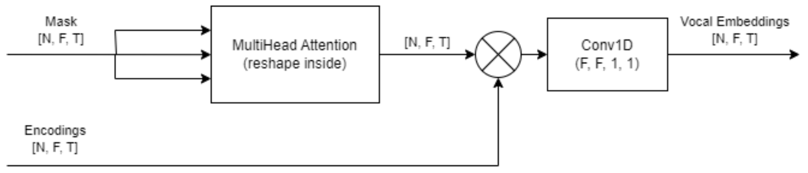

Mask [N, F, T]
MultiHead Attention (reshape inside)
[N, F, T]
Conv1D (F, F, 1, 1)
Vocal Embeddings [N, F, T]
Encodings [N, F, T]

Figure 5: Decoder Layer

6. Output Layer
   Output layer is the reverse operation of embedding layer. Firstly there is a convolution. And if it's the spectrogram, a ISTFT is applied.
7. Loss Function for Separation
   We use -SI-SNR(Scale-Invariant Signal-To-Noise Ratio) as loss function. The "scale invariant" means the strength(or amplitude, power) is unconcerned to compare the similarity of 2 signals. But it also has a problem, that the output music is often louder than original source. This is why we need a scale layer to fix that. We use SI-SNR for 1st stage training (train separation), and 2nd stage we train the scale layer to ensure not only the signal is similar, but has the same loudness. Si-SNR is defined as follows:

$$s_{target} = \frac{< \hat{s}, s > s}{\|s\|^2}$$
$$e_{noise} = \hat{s} - s_{target}$$
$$SISNR = 10log_{10} \frac{\|s_{target}\|^2}{\|e_{noise}\|^2}$$

8. Loss Function for Scaling
   We train a simple linear network in 2nd stage to scale the output to match the input signal. In this stage we use -SDR metrics as loss.

$$SDR = 10log_{10} \frac{\|s\|^2}{\|s - \hat{s}\|^2}$$

9. Two-Way Architecture
   Finally, we combine the waveform architecture and spectrogram architecture together, the output waveforms are added to calculate losses.

5 Experiments and Results

Because the previous work (Att-Tasnet) is for speech separation, our datasets and areas are different. So we don't compare the results with previous work, instead we will compare the results of time-domain, frequency-domain and hybrid approaches.

1. Hyperparameters and Tuning
   See Hyperparameter Table for definition. These hyperparameters can be grouped into different categories:

Parameter	Description	Sample Value
N	Batch size	512
lr	Learning rate	1e-4
lr_sched	LK scheduler config	Linear
L	audio segment length	16000
W	window size	512
H	hop size	160
F	number of features	512
NH	numher of heads	16
Local	Local attention width	20
Dropout	Dropout rate	0
B	TCN bottleneck dim	128
BC	TCN block channel	512
D	TCN dilation number	4
R	TCN repeat number	3
BH	TCN block channel num	1024

Table 1: Hyperparameters

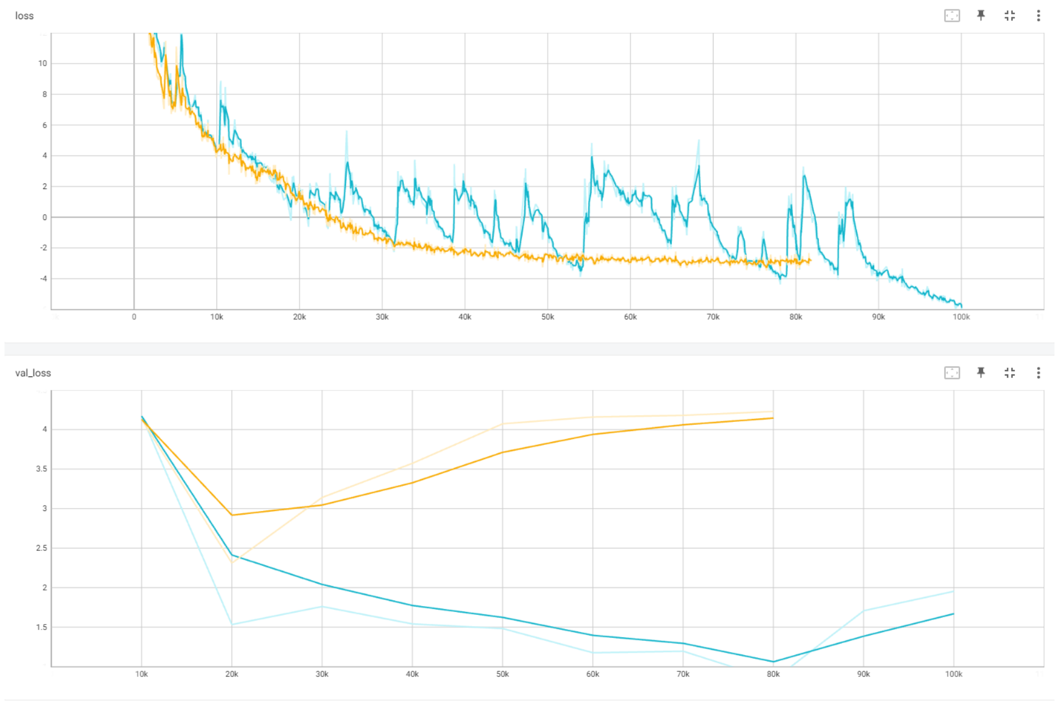
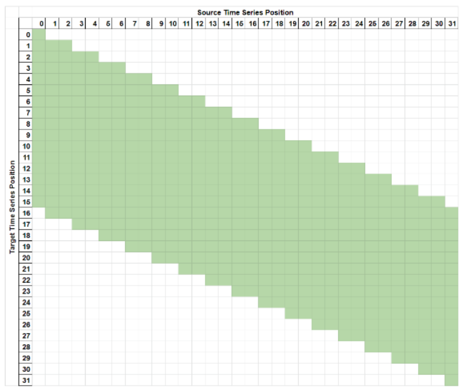
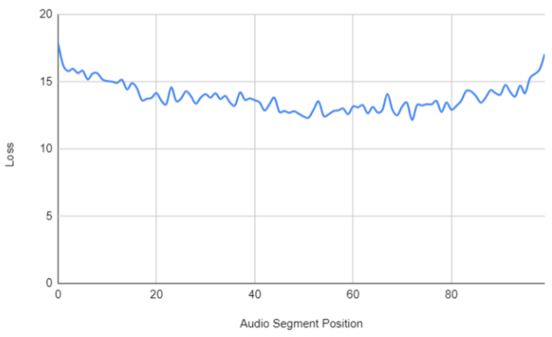


Figure 6: Learning Rate and Loss Curve

- Learning speed
  N,lr,step,gamma determine the learning speed and quality.
  N also affects the running memory. So we make N as large as possible as long as memory is enough. LR affects the learning speed and quality. Too small lr makes the learning slow, too large makes it don't converge. We firstly train with a randomly selected LR and analysis. When it's 1e-4, the training loss oscillate a lot, this shows the LR is too large in the oscillation period. Then an StepLR(step, gamma) scheduler is applied, after every 'step' steps, lr = lr * gamma. We try (50, 0.7) over 1000 epochs. The loss curve is much smoother, but the validation loss is much higher than previous one in later period. This shows it may be trapped in a basin on a plateau(Figure 6. See appendix for more LR tuning details.



A - Local finder mask



B - Loss by position

Figure 7: Local Finder Method

- Receptive field
  W, F, Local, D determine the receptive field of the algorithm. The receptive field in our problem is like: how long neighbors should I care about to separate this audio segment? It's 0.025ms, 0.05ms or 0.1ms? The right receptive field make the algorithm perform best, the larger will cost longer time to train, the smaller will lose performance.
  We find the best "Local" by a non-uniform local mask method: the local size of position close to both sides is smaller and the center position is larger as Figure 7-A. And train with this mask, compare the loss position by position. The loss of middle position should be lower than both sides. The result is like Figure 7-B.
  We can find that the loss of middle part is indeed smaller, but the middle 20% is not remarkably better than 20% to 80% range. The local_size of 20% position is just 20, so we choose this number. Also note that, this local_size is decided with wave length, sample rate, W and F. Since the segment is 1s duration, we can say the neighbor 0.5s contribute most to the position. So if we change wave length or sample rate, the local_size should be changed correspondingly.
- Fitting degree
  F, B, R, BH determine the fitting degree. The larger, the model is deeper, it has more parameters to fit but requires more time to train, and more likely to overfit. We just choose a set of proper initial values, and increase them gradually when iterate.
- Layer Normalization Type There are 3 types of layer norm: No layer norm(magenta), 1D layer norm(purple) and 2D layer norm(yellow). We can tell from Figure 8, without layer norm, the training is faster, but also easy to overfit. With a 2D layer norm, the training is slowest, the val_loss is not good either. I guess this is because some key information is lost due to the normalization. The 1D layer norm is the best, the val_loss is the lowest.

2. Time, Frequency and 2-Types of Hybrid Methods
   We have 2 hybrid methods. One is concatenate waveform embedding and spectrogram embedding as new audio embedding. The other is hybrid both architecture, add the outputs as the final output.
   We also train these algorithms in different scale of nets, the 1st one is relatively smaller, the 2nd one is larger to see the differences of performance.

3. Results
   - Smaller Net
     See Figure 9 for hyper parameters and loss charts. The best results of each algorithm are: waveform train(-1.07) val(0.825), spectrogram train(1.367) val(1.398), hybrid embedding train(0.540) val(0.866), hybrid architecture train(-2.661) val(0.783).

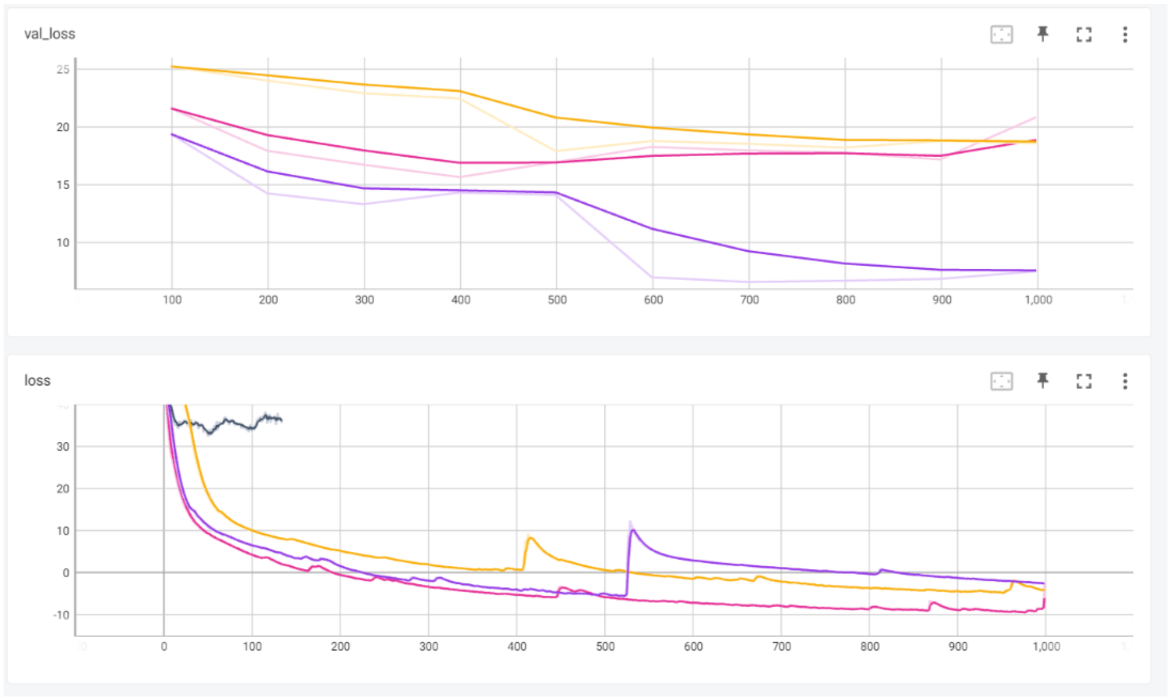


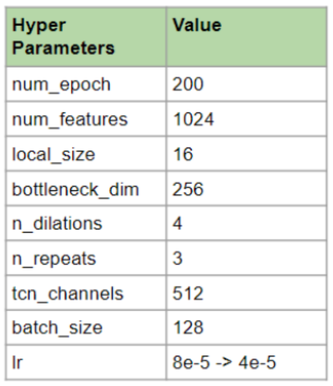
Figure 8: Different Layer Norm in Embedding Layer(magenta - none, purple - 1D, yellow - 2D)

Performance: Hybrid Architecture > Waveform Embedding > Hybrid Embedding > Spectrogram Embedding.
Stability(difference between train_loss and val_loss): Spectrogram Embedding > Hybrid Embedding > Waveform Embedding > Hybrid Architecture.

- Larger Net
  See Figure 10 for hyper parameters and loss charts. The best results of each algorithm are: waveform train(6.71) val(5.03), spectrogram traintrain(5.67) val(4.52), hybrid embedding train(4.46) val(3.93).
  Performance: Hybrid Embedding > Spectrogram Embedding > Waveform Embedding.
  Stability(difference between train_loss and val_loss): Hybrid Embedding > Spectrogram Embedding > Waveform Embedding.

6 Conclusion

1. Conclusion of fine tuned smaller network
   (a) Waveform Embedding(Baseline)
      - Parameters directly contribute to the output, good for fitting, but easier to overfit.
   (b) Hybrid Architecture
      - Doubling the parameters, performance not getting better remarkably.
   (c) Spectrogram Embedding
      - Parameters don't directly contribute to the output (they have to go through ISTFT first).
      - Very good characteristic extraction! The difference between val loss and test loss is the least of all!
      - But the performance is relatively worst. (Some original information is lost when applying STFT and ISTFT).
   (d) Hybrid Embedding
      - Most robust and good enough approach!



Hyper Parameters	Value
num_epoch	200
num_features	1024
local_size	16
bottleneck_dim	256
n_dilations	4
n_repeats	3
tcn_channels	512
batch_size	128
lr	8e-5 -> 4e-5

Hyperparamters

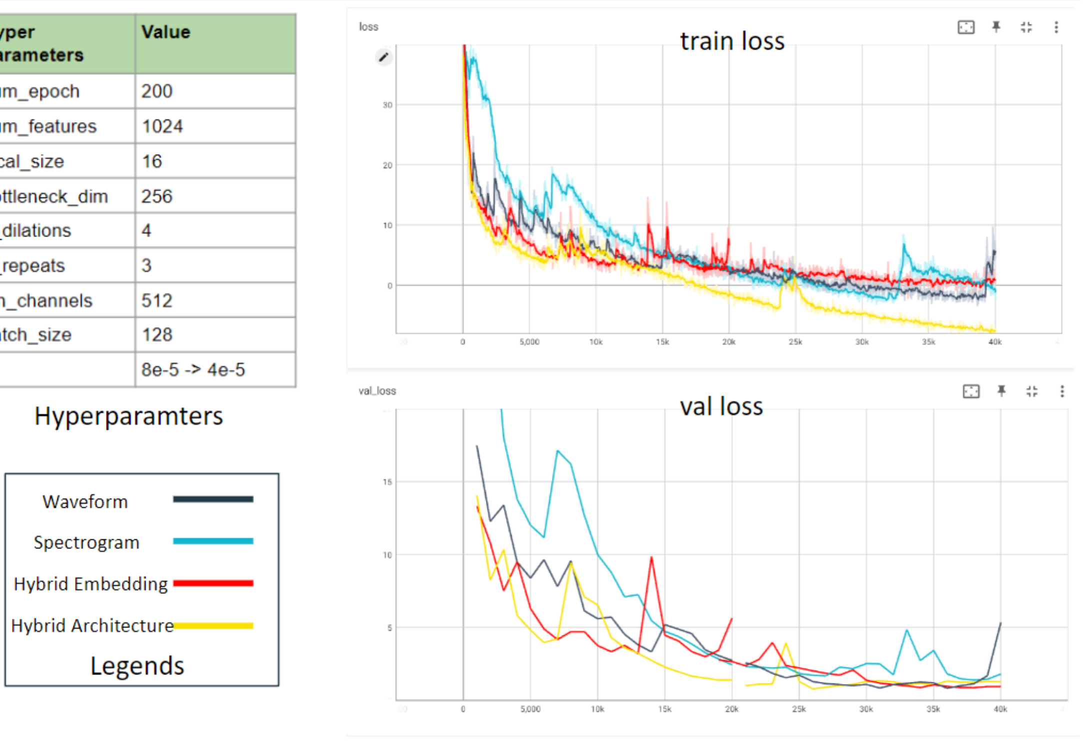


Figure 9: Smaller Net Results

- The spectrogram part is like a stablizer to prevent overfit and keeps tracking the musical characteristics.
- The waveform part seems fixing the details where the spectrogram may lose.

2. Conclusion of a expanded network
   (a) Waveform Embedding(Baseline)
      - The performance rapidly decays, it becomes the worst of all.

   (b) Spectrogram Embedding
      - As stable as before. The performance gets better than waveform embedding.

   (c) Hybrid Embedding
      - Best of the three. Both stable and well performing.

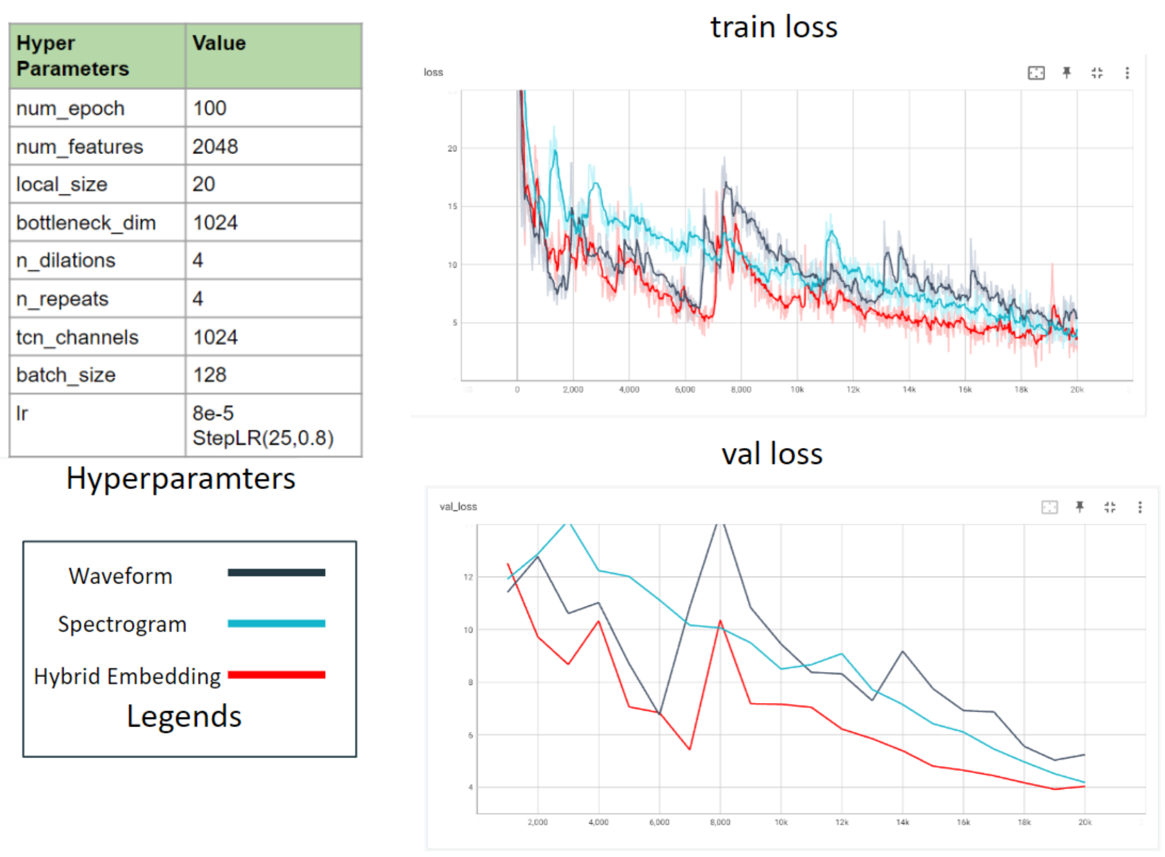
3. Conclusion of audio Hybrid Embeddings
   A very promising audio data representation.
   - It balances musical expressions and audio waveform details.
   - It can prevent overfitting better, suitable for very large architecture.
   - At the same time, the performance is good as well.

7 Future Work

These experiments didn't reach sota, more tuning on architecture details is required, such as different types of normalization, activation functinos, adding more layers to encoders/decoders. Also, we should find better LR scheduler. At last, we will dig deeper into Hybrid Embeddings.

8 Contributions

All codes of model/data/train are the original work of Jiabin Wang. The algorithms of attention en-decoder and TCN parts are based on Att-Tasnet Paper (4). The metrics of SDR are from Hybrid Paper (1). The libraries we use include PyTorch(7), PyTorch-Lightning(8).



Hyper Parameters	Value
num_epoch	100
num_features	2048
local_size	20
bottleneck_dim	1024
n_dilations	4
n_repeats	4
tcn_channels	1024
batch_size	128
lr	8e-5 StepLR(25,0.8)

Figure 10: Larger Net Results

9 Appendix - Hyperparameters Tuning Details

Please see Figure 11.